Vol. 6, No. 3: 00–00

Consistent Subdivision of Convex Polyhedra into Tetrahedra

Nelson Max
Lawrence Livermore National Laboratory, and University of California, Davis

Abstract. This paper presents a simple method of subdividing a grid of convex polyhedral cells into tetrahedra such that the subdivisions of two adjacent cells divide their common face into the same set of triangles. The method is then generalized to grids of convex polytopes in n dimensions.

1. Introduction

Visualization algorithms for unstructured polyhedral grids are often simplified if the grid cells are subdivided into tetrahedra. For example, the cell projection method is easier for tetrahedra [Shirley, Tuchman 90] than for general polyhedra [Max et al. 00]. It is easier to generate a contour surface for tetrahedral grids than for grids with cubes or more general cells. The tetrahedral subdivision of a cell produces a piecewise linear interpolation, which is simpler than the piecewise polynomial forms for the interpolation of vertex values across more general cells.

It is important that the subdivision be "consistent" in the sense that the subdivision of two adjacent cells divides their common face into the same set of triangles to guarantee that the piecewise linear functions agree on the face and jointly define a continuous function. More generally, an n-dimensional grid is called *consistent* (or *conforming*) when any two intersecting cells in the grid intersect along a set of common faces. This paper gives a subdivision algorithm that guarantees this consistency when the input grid is consistent.

1086-7651/01 $0.50 per page

Nielson and Sung [Nielson, Sung 97] gave a method for grids whose cells have the combinatorial topology of tetrahedra, cubes, triangular prisms, or "crystals." In Max et al. [Max et al. 00] we give a similar method for grids whose cells have the combinatorial topology of tetrahedra, cubes, triangular prisms, or square pyramids. Both of these subdivision methods, as well as the one proposed here, use a linear ordering of the vertices. The most common grid data structure lists the coordinates and data values for the grid vertices in large arrays, and each cell is defined by its topological type and a list of its vertex indices. The ordering of the integer indices then provides the desired linear ordering of the vertices.

2. Algorithm

The algorithm for a grid of three-dimensional polyhedra starts with the trivial conversion of the descriptions of the vertices and edges from cell data types to simplex data types. It then subdivides all the two-dimensional face polygons into triangles. Finally, it subdivides all the polyhedra into tetrahedra.

Input: A cell complex of dimension three, that is, a collection of convex cells of dimension $k \leq 3$. The 0-cells are the vertices, the 1-cells are the edges, the 2-cells are the faces, and the 3-cells are the polyhedra. Each k-cell is defined by the list of its boundary face cells of dimension $k-1$.

Output: A simplicial complex, that is, a cell complex where all cells are simplices. A k-simplex is defined by its list of $k+1$ vertices. Thus the 2-cells are all triangles, and the 3-cells are all tetrahedra.

Step 0: Reformat the 0-cells into equivalent simplex descriptions.

Step 1: Reformat the 1-cells into equivalent simplex descriptions.

Step 2: Divide each 2-cell F into triangles, by drawing diagonals from the least index vertex V of F. These triangles join V to each edge of F that does not have V as a vertex.

Step 3: Divide each 3-cell K into tetrahedra by joining its vertex V of least index to all the triangles from faces F in the boundary of K that do not have V as a vertex.

To show consistency, we need to show that the subdivision of K in Step 3 is consistent with the subdivision of its faces in Step 2. This is clearly true along a face F not having V as a vertex, because the tetrahedra were formed from the triangles in F. For a face G having V as a vertex, V is the least index vertex in G, since it is the least index vertex in all of K. Therefore

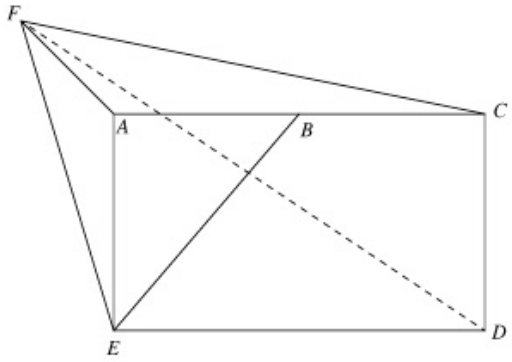

Figure 1. Quadrilateral pyramid $ABFE$ and triangular prism $BCDEFG$, both shown in thick lines, share face $BCFE$. The vertex indices increase in alphabetical order. Thin lines show the resulting tetrahedra $ABCF, ABFE, BCDG, BCGF$, and $BEFG$, which meet consistently, in particular, along the two triangles BCF and BFE into which face $BCFE$ was subdivided.

Step 2 divides G by diagonals from V. This is consistent with the way the tetrahedra joined to V in Step 3 subdivide G. See Figure 1 for an example.

3. Degenerate Cases

For the algorithm to be useful, the input cells must be *strictly convex*, that is, convex and with the further property that if C is a grid cell, F is a lower-dimensional k-face of C, and H is the hyperplane of dimension k containing F, then no other vertex of C (except those of F) lies in H. Figure 2 shows a case where this condition is violated and some of the tetrahedra constructed are degenerate. The piecewise linear function defined by interpolation inside the nondegenerate tetrahedra may then have discontinuities across the degenerate tetrahedra.

On the other hand, if K is strictly convex, the tetrahedra constructed from it in Step 3 will be nondegenerate and have disjoint interiors. They will stay within K and together fill K. Therefore, the piecewise linear interpolation will be well defined and continuous.

Polyhedra in grids with nonplanar faces cannot be convex, but the algorithm can be viewed as providing a collection of tetrahedra that join the grid vertices consistently and fill the grid volume. The motivating application is to selectively subdivide cells with nonplanar faces in a view-dependent manner in order to break cycles in a visibility sort for back-to-front compositing. (See [Max et al. 00].)

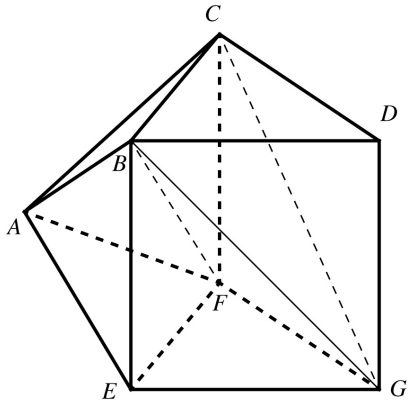

Figure 2. A nonstrictly convex polyhedron $ABCDEF$, with edges AB and BC along the same line, and with triangular face ABE and quadrilateral face $BCDE$ in the same plane. The vertex indices increase in alphabetical order. The algorithm will divide face $ABCF$ into degenerate triangle ABC and nondegenerate triangle ACF, and face $BCDE$ into triangles BCD and BDE. It will produce two degenerate tetrahedra $ABCD$ and $ABDE$, as well as two nondeteriorate tetrahedra $ACDF$ and $ADEF$.

4. Generalization to n Dimensions

The input to the n-dimensional algorithm is a cell complex of dimension n, that is, a collection of cells of dimension $k \leq n$. Each k-cell is defined by the list of its boundary faces of dimension $k-1$. These faces must fit together with the manifold topology of a $(k-1)$-sphere. However, the whole cell complex need not have manifold topology, and the subdivision algorithm will still work for nonmanifold cell complexes.

In an abstract cell complex, the 0-cells (vertices) carry no information except their unique names, but in a geometric cell complex, they have coordinates in some m-dimensional space. The algorithm below applies to abstract cell complexes, since it does not use any vertex geometry. However, the concept of consistency does not have meaning unless the geometry of the vertices is specified, so that the intersection of two different cells makes sense.

The construction in n dimensions follows the same pattern as in three dimensions, with Step i replacing the cells of dimension i with i-simplices. Thus Step 2 replaces 2-cells (polygons) by 2-simplices (triangles), and Step 3 replaces 3-cells (polyhedra) by 3-simplices (tetrahedra). The construction `Join(V,F)`, joining a vertex `V` to an i-simplex `F`, returns an $(i+1)$-simplex with all the vertices of `F`, plus one added vertex `V`.

```
simplicial-complex Subdivide(cell-complex U) {
   n = dimension U
   for (i = 0; i <= n; i++) {
      U = Step(U, i)
   }
   return U;
}

cell-complex Step(cell-complex U, int i){
   for each cell C of dimension i
      if (i <= 1)
         replace C by (simplex) C
      else {
         V = vertex of C with lowest index
         for each face F of C of dimension i-1
            if (F does not have V as a vertex) {
               J = Join(V, F)
               add J to U
            }
         remove C from U
      }
   return U
}
```

5. Consistency Proof

The output in n dimensions is consistent for basically the same reasons as in the three-dimensional case, but a precise proof takes somewhat more work. The pseudocode above produces the correct list of n-simplices at the end of `Subdivide`, but the intermediate results are somewhat inconsistent, because `Step` does not add all the faces of the new simplices it adds, and does not replace the cell `C` in the face lists of other cells when it removes `C` from `U`. We will use induction to prove that `Subdivide` produces a consistent grid of simplices when its input cell complex is consistent; we need to show that each `Step(U,i)` produces a consistent cell complex when its input is consistent.

The revised pseudocode below produces good cell complexes after every intermediate step, taking care to replace cell `C` in the face lists of the other cells when it removes `C` from `U`. Since `Step(U,i)` replaces only the cell descriptions in dimension i with simplex descriptions, the intermediate grids are hybrids of cells and simplices. This is not a problem, since the boundary faces of a simplex `C` can be constructed easily from its vertex list. The input of `Step(U,i)` is a hybrid structure with all cells of dimension $j < i$ replaced by simplices, and the output is one with all cells of dimension $j \leq i$ replaced by simplices.

The construction `AllBoundary(C)` follows the boundary lists of the faces of cell `C` recursively to build the set of all faces of any dimension in the boundary of `C`. The loop over the dimension `j` of the cells in `AllBoundary(C)` ensures that when a simplex is added to `U`, its faces are already in `U`. The cell sets `T, X,` and `Y` are not needed in the algorithm, but are used in the consistency proof.

```
hybrid-complex Step (hybrid-complex U, int i){
  for each cell C of dimension i
    if (i <= 1)
      replace C by (simplex) C
    else {
      V = vertex of C with lowest index
      cell set S = {} // for replacing C in boundary lists
      cell set X = {}, Y = {}, T = {} // for proof only
      D = AllBoundary(C)
      for (j = 0; j <= i, ++j) {
         for each face F in D of dimension j
            if (F does not have V as a vertex) {
               J = Join(V, F)
               if (J is not in U) {
                  add J to U // the faces of J are already in U
                  add F to Y
                  if (j == i) add J to S
               }

               else
                  add F to T
            }
            else
               add F to X
      }
      for every cell E in U of dimension i+1
         if(C appears in the boundary list of E)
            replace C by S in the boundary list of E
      remove C from U
      }
   return U
}
```

Suppose the input U_0 to `Subdivide` is a consistent geometrical cell complex with strictly convex cells. We prove that the output is consistent by induction on `i` in the `for` loop in `Subdivide`. If `i` is 0 or 1, `Step(U,i)` does not change the geometry of the grid; it only replaces cell descriptions by simplex descriptions, so the output is still consistent. Now we assume that the input

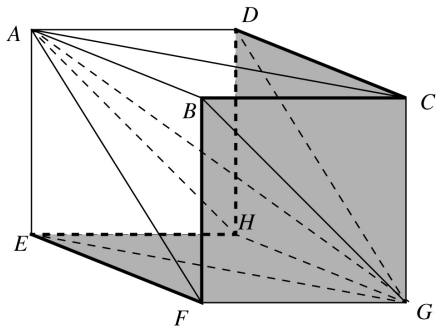

Figure 3. Hexahedron $ABCDEFGH$, considered as a 3-cell C subdivided by the algorithm. Vertex indices are increasing in alphabetical order, so A is the least index vertex V. The set T consists of the vertices B, C, D, H, E, and F, and the edges BC, CD, DH, HE, EF, and FB, which are drawn with thick lines. The set Y is drawn lightly shaded, and consists of vertex G, edges BG, CG, DG, HG, EG, and FG, and triangles BCG, CDG, DHG, HEG, EFG, and FBG.

to `Step(U,i)` is a consistent hybrid complex, and we show that its output is also consistent. It is sufficient to show that the subdivision of a single cell `C` is consistent with itself in the interior of `C`, and is consistent with the subdivision of the boundary of `C`, because then consistency between (pieces of) different cells follows from the consistency of the input to `Step(U,i)`.

Let `C` be an i-cell, let `V` be the least index vertex of `C`, and let `D = AllBoundary(C)`. Let `X` be the subset of `D` of faces with `V` as a vertex; let `T` be the subset of `D` of the faces `F` such that `Join(V,F)` is in `U`, and let `Y = D-X-T` be the remaining faces in `D`. (See pseudocode above and Figure 3.) Now, if `B` is a cell of $\mathtt{U}_0$ of any dimension that is in the boundary of `C`, and `B` has `V` as a vertex, then `V` is also the smallest index vertex of `B`, since the vertices of `B` are a subset of those of C. Thus all these cells have been subdivided using joins from `V`, to produce cells in `X`. The other cells of $\mathtt{U}_0$ in the boundary of `C`, without `V` as a vertex, produce cells in `T` or `Y`. Note that a simplex `J = Join(V,F)` is added to `U` by `Step(U,i)` acting on `C` if and only if `F` is a member of `Y`. Each such face `F` comes from a face $\mathtt{F}_0$ in $\mathtt{U}_0$ not containing `V` as a vertex, since all faces with `V` as a vertex were used to build `X` instead. Therefore, since $\mathtt{U}_0$ is strictly convex, `V` is not in the plane of $\mathtt{F}_0$ or `F`, and each added simplex `J` is nondegenerate. (We need to go back to $\mathtt{U}_0$ here, because `Step(U,i)` produces grids which are not strictly convex when it subdivides faces.) Again, by convexity, the interiors of all such added cells are disjoint, lie inside `C`, and fill the interior of `C`. Since `Join(V,F)` $\cap$ `Join(V,G)` = `V` $\cup$ `Join(V,F`$\cap$`G)`, and by the induction hypothesis on the input to `Step(U,i)`, `D` is consistent, and the added cells are consistent among themselves.

Recall that the cells of $\mathtt{U}_0$ in the boundary of C that have V as a vertex have been subdivided using joins from V. This means that T is the boundary of Y, and X is V plus the cells of the form Join(V,F), with F in T. Thus the cells added to replace C are consistent with the part of D in $\mathtt{X} \cup \mathtt{T}$. Finally, these added cells are consistent with the part of D in Y by their construction as Join(V,F) for F in Y. This shows that the subdivision of C is consistent with D.

Acknowledgment. This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract number W-7405-ENG-48, and was specifically supported by the Accelerated Strategic Computing Initiative. Suggestions from jgt editors Ronen Barzel and John Hughes greatly improved this paper.

References

[Max et al. 00] Nelson Max, Peter Williams, and Claudio Silva, Approximate Volume Rendering for Curvilinear and Unstructured Grids by Hardware-Assisted Polyhedron Projection, Vol 11 (2000) pp. 53 - 61.

[Nielson, Sung 97] Gregory Nielson and Junwon Sung, Interval Volume Tetrahedralization, Proceedings of IEEE Visualization 97, pp. 221 - 228.

[Shirley, Tuchman 90] Peter Shirley and Alan Tuchman, "A Polyhedral Approximation to Direct Scalar Volume Rendering," *Computer Graphics* 24:5 (November 1990) pp. 63 – 70.

Web Information:
http://www.acm.org/jgt/papers/Max01.

Nelson Max, L-560, Lawrence Livermore National Laboratory, 7000 East Avenue, Livermore, CA 94550. (max2@llnl.gov)

Received August, 2001; accepted in revised form December 5, 2001.